

<https://github.com/martensonbj/fem-intro-to-go>



(Hi! Fire this GitHub repository up in your browser for reference 🤘)

INTRO TO GO

FROM THE OTHER SIDE

(A LOVE STORY)

01. MEET CUTE

THE SETUP

WHY GO AT F.E.M?

(For real...you're going to need this)



ABOUT ME

Brenna Martenson



@martenson_bj

(gopherize.me)



highwing

WHAT TODAY LOOKS LIKE

- ▶ Brief Introduction
- ▶ Installation
- ▶ Basic Syntax & Structure
- ▶ Build some Go apps
- ▶ Brief look at concurrency

SLIDES

What information is available on the slides?

01. SETUP: SLIDES

01_folder/file.md
01_folder/code/file.go

SECTION NUMBER + BREADCRUMBS

FILES I AM REFERENCECING

(green means something actionable!)

(grey means lecture notes!)

SLIDES

CURRENT SECTION TITLE

(the repo structure will match the section structure)

HISTORY

BEFORE GO

It was 2007

It was Google

It was C++

Performance and scalability were hot topics

(Unsurprisingly, Google had become a large, difficult to maintain codebase)

- ▶ 1. Fast compile times
- ▶ 2. Ease of development
- ▶ 3. Fast execution

ENTER: GO

- ▶ Fast compile time
- ▶ Lots in common with C
- ▶ Reduces complexity of C
- ▶ Wicked fast build time
- ▶ Lightweight type system
- ▶ Concurrency
- ▶ Automatic garbage collection
- ▶ Strict dependencies
- ▶ Convention

INSTALLATION

THINGS YOU NEED

- ▶ An IDE of some kind (I'll be on VSCode)
 - ▶ Option Two: Go Playground
- ▶ A Terminal Window
- ▶ Your Favorite Browser (I'll be in Chrome)

Reminder: ^^ this file path lives here:
github.com/martensonbj/fem-intro-to-go)

(5 MINUTES)

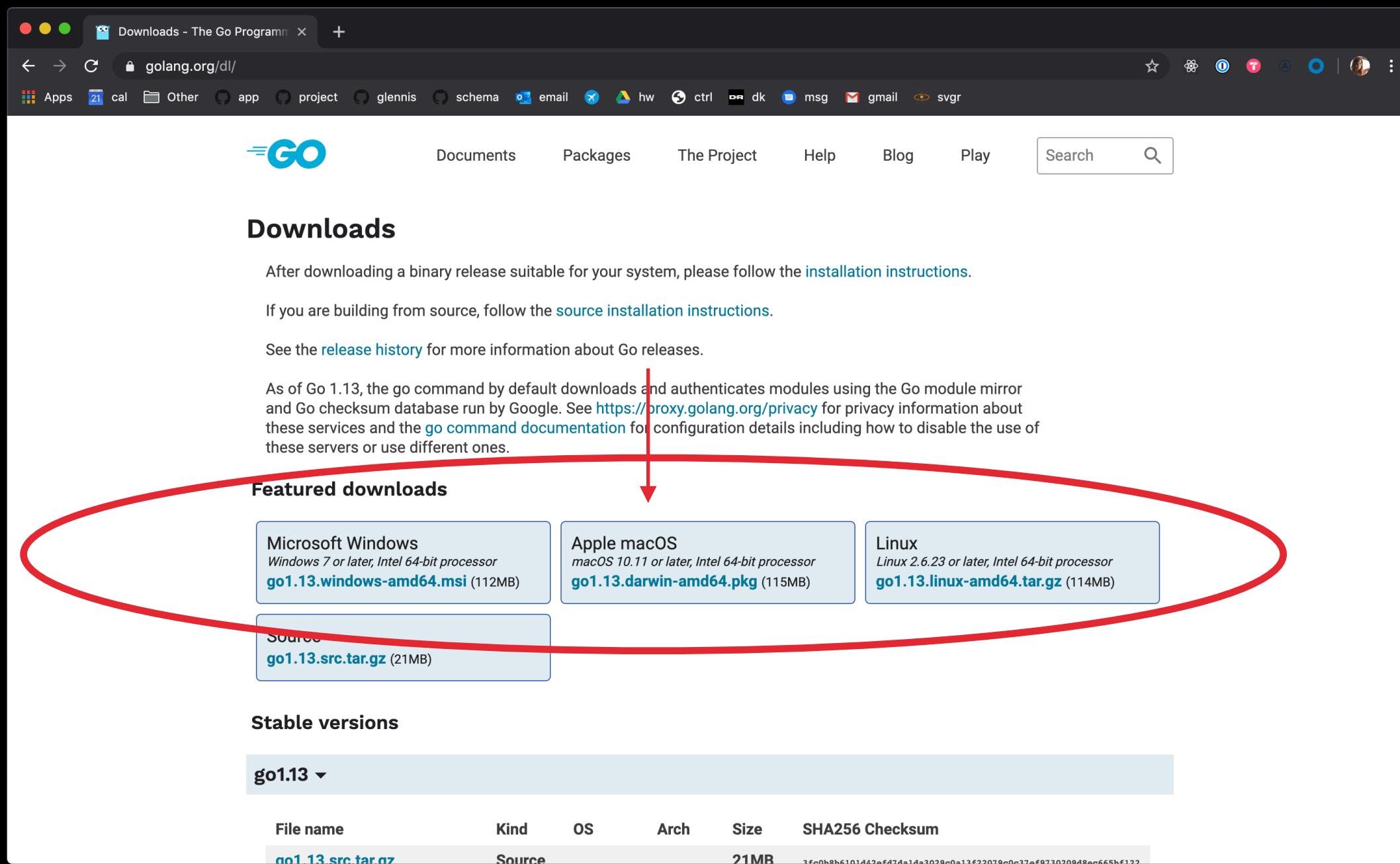
EXERCISE #0

INSTALLING GO

INSTALLING GO

▶ Install Go

▶ golang.org/dl



1. Verify Go was installed:

```
[brennamartenson] fem-intro-to-go [master] which go  
/usr/local/go/bin/go  
[brennamartenson] fem-intro-to-go [master] go version  
go version go1.13 darwin/amd64  
[brennamartenson] fem-intro-to-go [master]
```

2. Add these to your .bash_profile:

```
export GOPATH=$HOME/go  
export GOBIN=$GOPATH/bin  
export PATH=$PATH:$GOBIN
```

3. Verify those updates:

```
[brennamartenson] fem-intro-to-go [master] echo $PATH  
/usr/local/bin:/usr/bin:/bin:/usr/sbin:/sbin:/usr/local/go/bin:/usr/local/MacGPG2/  
bin:/Users/brennamartenson/go/bin:/Users/brennamartenson/go/bin  
[brennamartenson] fem-intro-to-go [master] echo $GOPATH  
/Users/brennamartenson/go
```

4. Create a workspace

- ▶ Navigate to your \$GOPATH (ie: /Users/brennamartenson)
- ▶ `mkdir go && cd go`
- ▶ `mkdir src && cd src`
- ▶ `git clone https://github.com/martensonbj/fem-intro-to-go.git`
- ▶ `cd fem-intro-to-go && go run main.go`

DOCUMENTATION

DOCUMENTATION & RESOURCES

The screenshot shows the official Go Programming Language website (golang.org) with a dark mode interface. The navigation bar includes links for Apps, cal, Other, app, project, glennis, schema, email, hw, ctrl, dk, msg, gmail, and svgr. The main menu features a logo with a stylized 'G', followed by 'Documents' (circled in red), 'Packages' (circled in red), 'The Project', 'Help', 'Blog' (circled in red), 'Play', and a search bar.

Go is an open source programming language that makes it easy to build simple, reliable, and efficient software.

A large cartoon gopher character is displayed below the text. A blue button labeled 'Download Go' with a cloud icon is present. Below the button, text indicates binary distributions are available for Linux, macOS, Windows, and more.

Featured articles

Go 1.13 is released
Today the Go team is very happy to announce the release of Go 1.13. You can get it from the [download page](#).
Published 3 September 2019

Module Mirror and Checksum Database Launched
We are excited to share that our module [mirror](#), [index](#), and [checksum database](#) are now production ready! The go

Try Go (with code editor)
Open in Playground ↗

```
// You can edit this code!
// Click here and start typing.
package main

import "fmt"

func main() {
    fmt.Println("Hello, 世界")
}
```

Hello, World! ▾ Run Share Tour

Featured video
GopherCon 2015: Robert Griesemer - ...

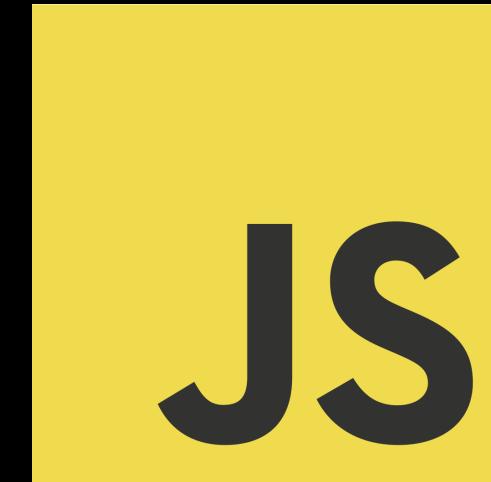
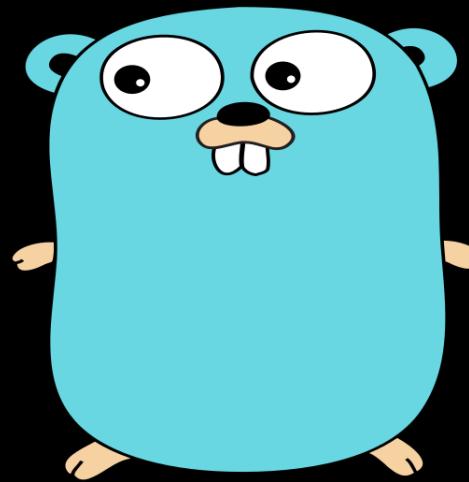
(5 MINUTES)

EXERCISE #1A

FIND STUFF

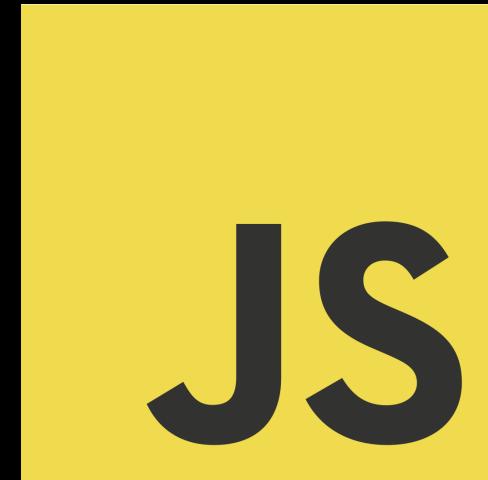
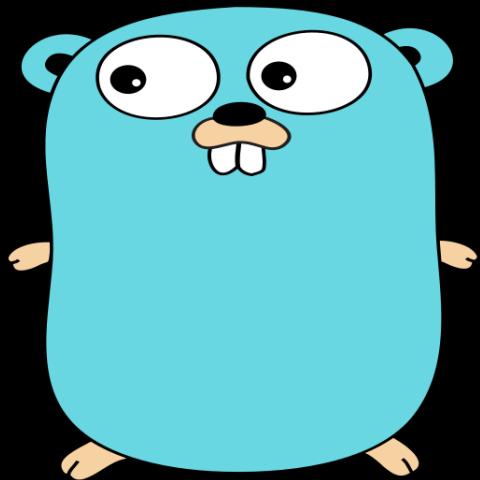
**02. FIRST DATES ARE
AWKWARD**
LETS GET THE WEIRD PARTS OUT OF THE WAY

TYPING



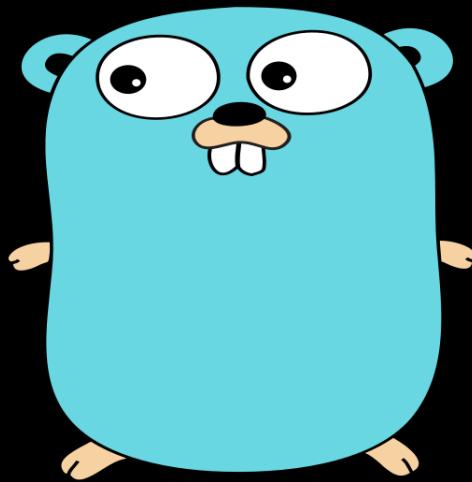
- ▶ Strongly typed
 - ▶ String, Float, Int, Byte, Struct...
- ▶ Dynamically typed
 - ▶ Variables can change
 - ▶ Typescript

STRUCTURES



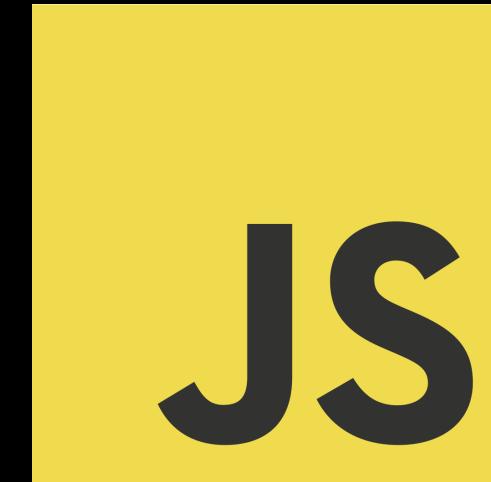
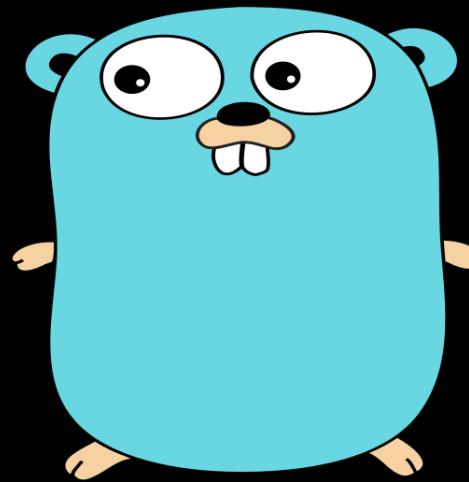
- ▶ Structs, Pointers, Methods, Interfaces
- ▶ Define behavior and attributes
- ▶ ES6 Classes (kind of)
 - ▶ Define behavior and attributes

ERROR HANDLING



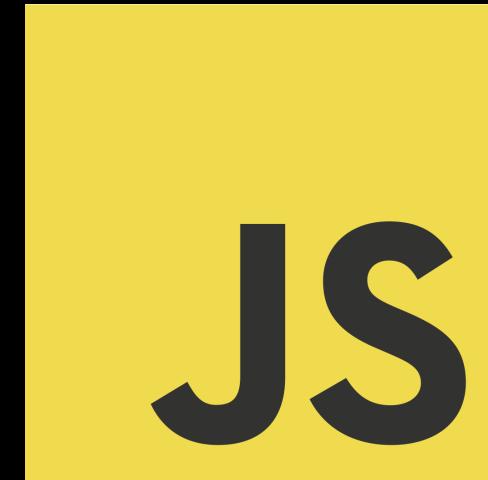
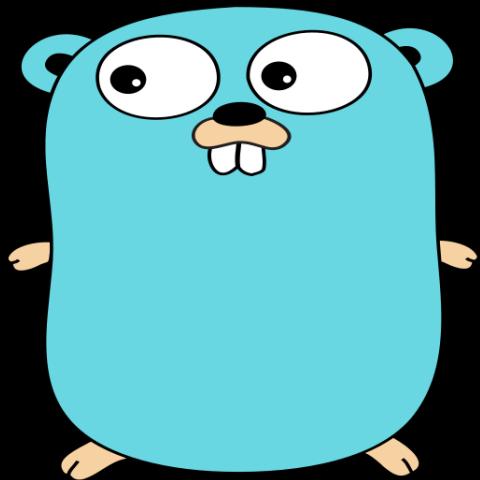
- ▶ Explicit
 - ▶ Sad path won't handle itself
-
- ▶ Built in
 - ▶ You'll get yelled at regardless

MULTI-TASKING



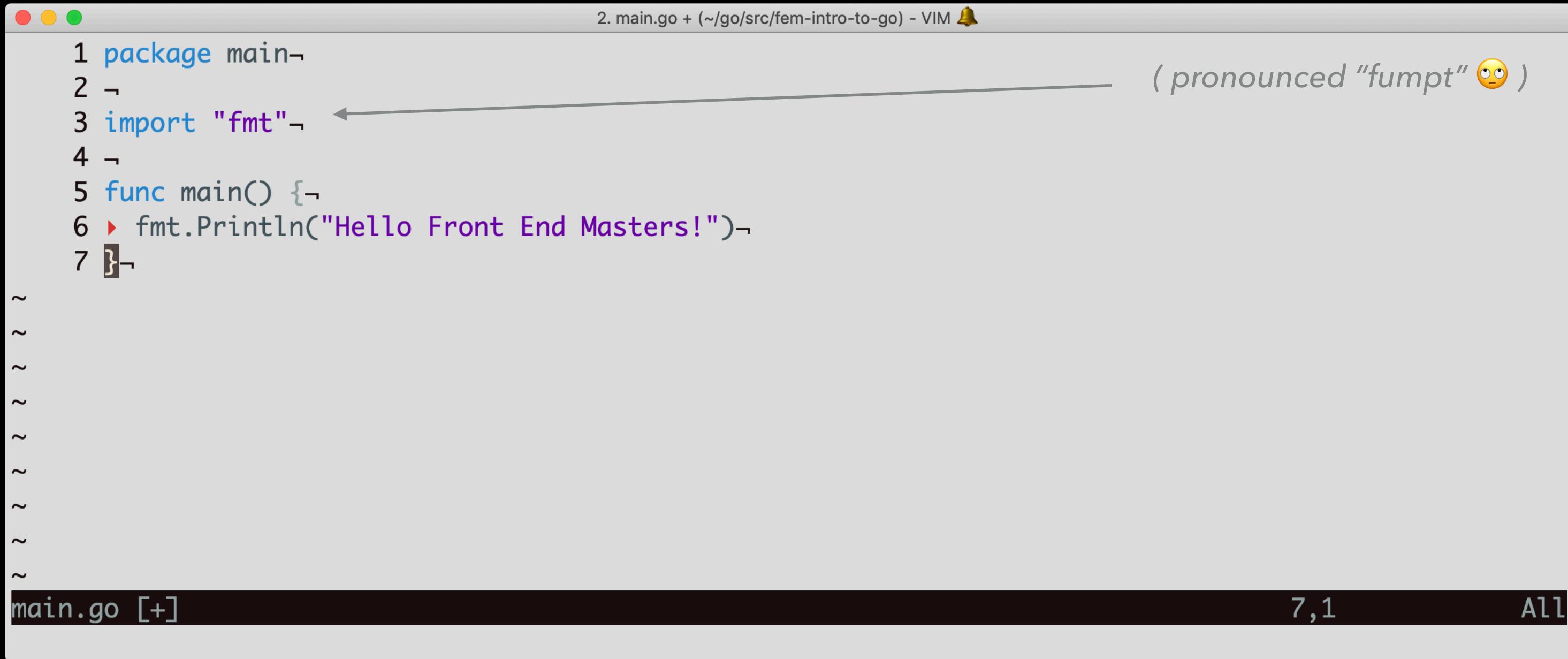
- ▶ Multi-Threaded
 - ▶ Concurrency,
Goroutines, Sync
-
- ▶ Single-Threaded
 - ▶ Callbacks, async await,
sagas, sadness

OPINIONATED-NESS



- ▶ Strong Opinions
- ▶ Convention, built in tooling and linters
- ▶ Fluid Opinions
- ▶ Subjective to the mood that day

ANATOMY (OF A FILE)



```
2. main.go + (~/go/src/fem-intro-to-go) - VIM 🎙
1 package main-
2 -
3 import "fmt"- ←
4 -
5 func main() {-
6   ▶ fmt.Println("Hello Front End Masters!")-
7 }
```

(pronounced "fumpt" 😕)

```
main.go [+] 7,1 All
```

PRINTING

`fmt.Println()`

PLAY.GOLANG.ORG

Goal: Experiment with printing

Print**fmt.Println()****fmt.Printf()**

- Prints output to the stdout console
- Returns number of bytes and an error
- (The error is generally not worried about)

Fprint**fmt.Fprintln()****fmt.Fprintf()**

- Prints the output to an external source (file, browser)
- Does not print to the stdout console
- Returns number of bytes, and any write errors

Sprint**fmt.Sprintln()****fmt.Sprintf()**

- Stores output on a character buffer
- Does not print to stdout console
- Returns the string you want to print

(5 MINUTES)

EXERCISE #2A

HELLO WORLD+

03. LET'S TALK

BASIC SYNTAX

TYPES

Name	Type Name	Examples
INTEGER	<code>int int8 int16 int32 int64 uint uint8 uint26 uint32 uint64</code>	<code>1 2 44 770 var age int = 21</code>
FLOAT	<code>float32 float64</code>	<code>1.5 3.14 2100 var gpa float64 = 4.0</code>
STRING	<code>string</code>	“Pancakes” <code>var plant string = “ficus”</code>
BOOLEAN	<code>bool && ! < <= >= == !=</code>	true false <code>var canDrink bool = age > 21</code>

PLAY.GOLANG.ORG

- ▶ Identify the type of a variable
- ▶ Convert types

VARIABLES

I'll be here:

03_BASIC_SYNTAX/CODE/VARIABLES.GO

But if you want, you can go here:

PLAY.GOLANG.ORG

CONTROL STRUCTURES

- ▶ If statements
- ▶ For loops
- ▶ Switch statements

IF STATEMENTS

03_BASIC_SYNTAX/CODE/IFS.GO

SWITCH STATEMENTS

03_BASIC_SYNTAX/CODE/SWITCH.GO

FOR LOOPS

03_BASIC_SYNTAX/CODE/FOR.GO

(7 MINUTES)

EXERCISE #3A

CONTROL STRUCTURES

04. OK BUT I WANT TO
KNOW MORE ABOUT YOU
MORE COMPLEX STRUCTURES

FUNCTIONS

```
func printAge(age int) int {  
    fmt.Println(age)  
    return age  
}
```

FUNCTIONS

04_COMPLEX_STRUCTURES/CODE/FUNCTIONS.GO

(5 MINUTES)

EXERCISE #4A

FUNCTIONS

VARIADIC FUNCTION

```
func doThings(args ...int) int {
    total := 0
    for _, num := range args {
        total += num
    }
    return total
}

func main() {
    fmt.Println(doThings(1, 2, 3))
}
```

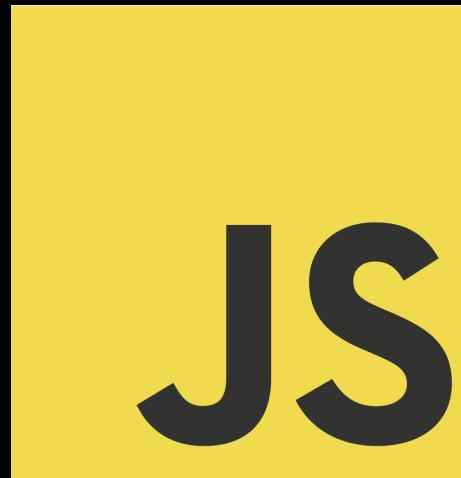
(5 MINUTES)

EXERCISE #4B

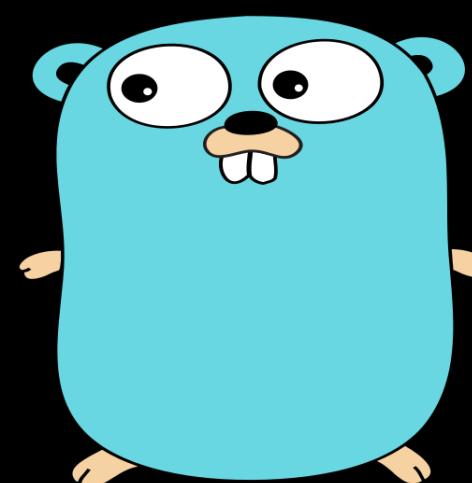
VARIADIC FUNCTIONS

ARRAYS

ARRAYS



```
1 // Initialize an empty array
2 const grabBag = []
3
4 // Eventually this array could have values that
5 // represent these types:
5 const grabBag = [string, int, boolean, float64]
6
7 // or
8 const grabBag = [string, string, string, string, integer,
8     boolean, float64]
9
```



```
7
8 // Initialize an empty array
9 var scores: [5]float64
10
11 // Eventually this array can ONLY contain floats and a
11 // length of 5:
12 [float64, float64, float64, float64, float64]
13
```

NOTE: Length is part of the type definition.
[5]float64 != [6]float64

ARRAYS

```
7
8 // Initialize an empty array
9 var scores [5]float64
10
11 // Eventually this array can ONLY contain floats and a
12 // length of 5:
13 [float64, float64, float64, float64, float64]
```

> _TRY IT_

> Copy line 9 from this example into the Go playground

> Print out the variable `scores` as is.

> What do you see?

ARRAYS: DEFINING VALUES

```
var scores [5]float64
scores[0] = 9
scores[1] = 1.5
scores[2] = 4.5
scores[3] = 7
scores[4] = 8
```

```
var scores [5]float64 = [5]float64{9, 1.5, 4.5, 7, 8}
```

```
scores := [5]float64{9, 1.5, 4.5, 7, 8}
```

```
scores := [...]float64{9, 1.5, 4.5, 7, 8}
```

> Try It

> Using range, iterate over the array of scores printing each value

> What error do you get?

ARRAYS: DEFINING VALUES

Needing to know the exact length of an array every time you need one seems problematic.

ENTER: THE SLICE

SLICES (+ MAKE)

Segments of an underlying array

Must be associated with space in memory

MAKE

According to the docs:

Make "Initializes and allocates space in memory for a slice, map, or channel."

```
1 package main
2
3 func main() {
4
5     var myArray [5]int
6     var mySlice []int
7
8     myArray[0] = 1
9     mySlice[0] = 2
10
11 }
```

04_complex_structures/code/slices.go

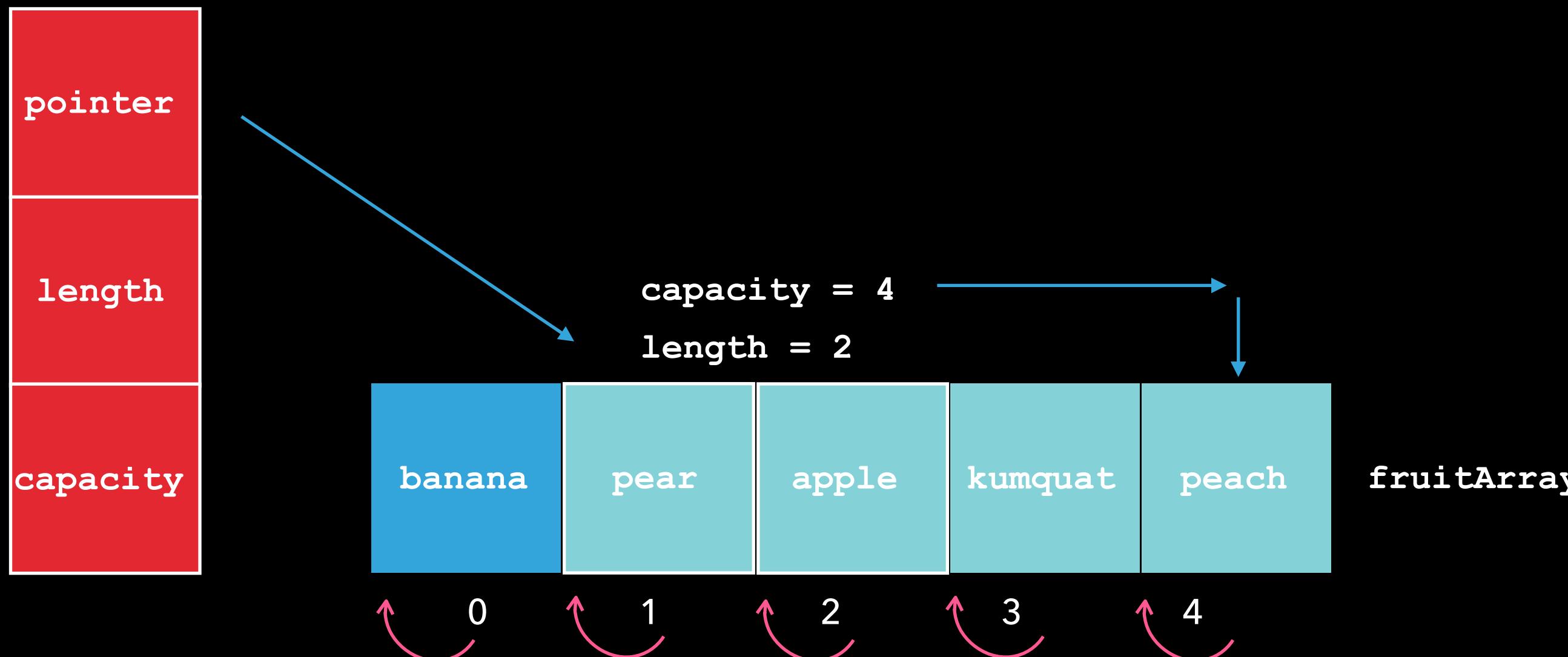
- > Try It
- > Print the results of each of these variables in your go program (or the go playground).
- > What happens?

SLICES

```
fruitArray := [5]string{"banana", "pear", "apple", "kumquat", "peach"}
```

```
var splicedFruit []string = fruitArray[1:3] // ==> ["pear", "apple"]
```

`splicedFruit`



SLICES

04_COMPLEX_STRUCTURES/CODE/SLICES.GO

- ▶ Modifying the length of a slice
- ▶ Append
- ▶ Copy

MAPS

MAPS

```
1 package main
2
3 import "fmt"
4
5 func main() {
6
7     var userEmails map[int]string
8
9     userEmails[1] = "user1@gmail.com"
10    userEmails[2] = "user2@gmail.com"
11
12    fmt.Println(userEmails)
13
14 }
```

> _TRY IT_

> Add this code into a go file or playground

> Add a third email and run the program

> What happens? What are we missing?

(7 MINUTES)

EXERCISE #4C

COMPLEX STRUCTURES: SUMMARY

05. SO WHERE ARE YOU FROM?

THE GO TOOLKIT & PACKAGES

GO TOOLS & COMMANDS

GO TOOLS & COMMANDS

go run main.go

go install

go build

go fmt main.go

go list

go vet

go doc fmt.Println

go get golang.org/x/lint/golint

golint

PACKAGES

```
package main  
  
import (  
    "fmt"  
    "math"  
    "reflect"  
)
```

PACKAGES

05_TOOLKIT/CODE/PACKAGES.GO

- ▶ Go packages
- ▶ Package visibility
- ▶ Custom packages

UNIT TESTING

(JUST A CASUAL GLANCE)

UNIT TESTING

`average.go`
`average_test.go`

```
5 package utils
6
7 import "testing"
8
9 func TestAverage(t *testing.T) {
10     expected := 4
11     actual := utils.average(1, 2, 3)
12
13     if actual != expected {
14         t.Errorf("Average was incorrect! Expected: %d, Actual: %d", expected, actual)
15     }
16 }
```

`go test`

(5 MINUTES)

EXERCISE #5A

TEST THE ADD METHOD

06. WHAT DEFINES YOU?

STRUCTS

STRUCTS

06_structs/code/structs.go

```
// User is a user type
type User struct {
    ID ..... int
    firstName string
    lastName string
    email ..... string
}
```

```
2
3 // User is a user type
4 type User struct {
5     ID ..... int
6     firstName, lastName, email string
7 }
8
```

```
11 func main() {
12     u := User{ID: 1, firstName: "Marilyn", lastName: "Monroe", email: "marilyn.monroe@gmail.com"}
13
14     fmt.Println(u)
15 }
```

(5 MINUTES)

EXERCISE #6A

WORK WITH STRUCTS

07. LET ME CHANGE YOU

POINTERS & REFERENCES

POINTERS

POINTERS*

```
1 package main
2
3 import "fmt"
4
5 func main() {
6     var name string
7     var namePointer *string
8
9     fmt.Println(name)
10    fmt.Println(namePointer)
11    fmt.Println(&name)
12 }
13
```

A **pointer** in Go is a variable that holds the **memory location** of that variable instead of a copy of its value.

POINTERS

Modifying Pointers

07_POINTERS/POINTERS.GO

```
1 package main
2
3 import "fmt"
4
5 func main() {
6     var name string
7     var namePointer *string
8
9     fmt.Println(name)
10    fmt.Println(namePointer)
11    fmt.Println(&name)
12 }
```

> _TRY IT_

> Set both `name` and `namePointer` to string values.

> What happens?

> What does the error message mean?

POINTERS: SUMMARY

- ▶ Pointer type definitions are indicated with a * next to the **type name**
 - ▶ Indicate that the variable will *point to a memory location*.

```
var namePointer *string
```

- ▶ Pointer variable values are visible with a * next to the **variable name**

```
var nameValue = *namePointer
```

- ▶ To *read through* a variable to see the pointer address
use a & next to the **pointer variable name**

```
var nameAddress = &namePointer
```

POINTERS

07_POINTERS/POINTERS.GO

- ▶ Pass by value
- ▶ Pointers & functions
- ▶ Pointers & structs

(5 MINUTES)

EXERCISE #7A

PRACTICE POINTERS

08. YOU F***** UP

ERROR HANDLING

ERROR HANDLING

ERROR

- indicates that something bad happened, but it might be possible to continue running the program.
- ie: A function that intentionally returns an error if something goes wrong

PANIC

- happens at run time
- something happened that was fatal to your program and program stops execution
- ex: Trying to open a file that doesn't exist

```
type error interface {  
    Error() string  
}  
  
err := someFuncThatReturnsAnError()  
  
fmt.Println(err.Error())
```

ERROR

08_ERRORS/CODE/ERRORS.GO

PANIC & DEFER

```
f, err := os.Open(filename)  
defer f.Close()
```

`panic(err.Error())`

08_ERRORS/CODE/ERRORS.GO

08_ERRORS/CODE/ERRORS.GO

RECOVER

- ▶ **Panic** is called during a run time error and fatally kill the program
- ▶ **Recover** tells Go what to do when that happens
 - ▶ Returns what was passed to *panic*.
- ▶ Recover must be paired with **defer**, which will fire even after a panic

09. METHODS

STATEFUL FUNCTIONS

METHODS

```
func (u *User) describe() string {
    desc := fmt.Sprintf("Name: %s %s, Email: %s, ID: %d",
        u.FirstName, u.LastName, u.Email, u.ID)
    return desc
}

func describeUser(u *User) string {
    desc := fmt.Sprintf("Name: %s %s, Email: %s, ID: %d",
        u.FirstName, u.LastName, u.Email, u.ID)
    return desc
}
```

09_METHODS.CODE/METHODS.GO

```
func main() {
    user := User{ID: 1, FirstName: "Marilyn", LastName: "Monroe", Email:
        "marilyn.monroe@gmail.com"}

    desc := describeUser(user)

    desc := user.describe()

    fmt.Println(desc)
}
```

(5 MINUTES)

EXERCISE #9A

PRACTICE METHODS

10. INTERFACES

A SET OF BEHAVIORS THAT DEFINE A TYPE

INTERFACES

```
type Mom struct {  
    FirstName: string  
    LastName: string  
    Kids: []Kid  
    BookClubFriends: []Friend  
}  
  
type Dad struct {  
    FirstName: string  
    LastName: string  
    Kids: []Kid  
    GolfFriends: []Friend  
}
```

```
var susan = Mom{...}  
var bob = Dad{...}
```

```
type Worrier interface {  
    CallFrequently([]Kid)  
    CheckIfOvenIsOff()  
}  
  
func (d Dad) CallFrequently(kids []Kid) {  
    fmt.Println("Clean your room")  
}  
  
func (m Mom) CallFrequently(kids []Kid)  
fmt.Println("Did you take your vitamins")  
}  
  
func (d Dad) CheckIfOvenIsOff() bool {}  
func (m Mom) CheckIfOvenIsOff() bool {}
```

susan is both type Mom and type Worrier
bob is both type Dad and type Worrier

"If it walks like a duck, swims like a duck and quacks like a duck, then it's a duck."

INTERFACES

Interfaces describe the kind of behavior our types can execute.

```
24 func (u *User) describe() string {
25     desc := fmt.Sprintf("Name: %s %s, Email: %s, ID: %d", u.FirstName, u.LastName, u.Email, u.ID)
26     return desc
27 }
28
29 func (g *Group) describe() string {
30     desc := fmt.Sprintf("The %s user group has %d users. Newest user: %s, Accepting New Users: %t", g.role, len(g.users), g.newestUser.FirstName, g.spaceAvailable)
31     return desc
32 }
33
34 func describeHumans(human Describer) string {
35     return human.describe()
36 }
37
38 func (human Describer) describe() string {
39     return human.describe()
40 }
```

THE EMPTY INTERFACE

```
interface{ }
```

- ▶ Specifies zero methods
- ▶ An empty interface may hold values of any type
 - ▶ These can be used by code that expects an unknown type
- ▶ Allows you to call methods and functions on types when you aren't entirely sure what will be expected
- ▶ Think the **any** type in Typescript

THE EMPTY INTERFACE

```
type User struct { }
```

```
type Admin struct { }
```

```
type Parent struct{ }
```

```
interface{ }
```

```
var people map[string]interface{ }
```

```
people = map[string]interface{
    "user": User,
    "admin": Admin,
    "parent": Parent,
}
```

11. WEB SERVERS

BUILDING A TODO LIST

ROUTES

“net/http”

```
func main() {  
    http.HandleFunc("/", home)  
}
```

11_ROUTES/CODE/ROUTES.GO

(20 MINUTES)

EXERCISE #11A

BROWSER TODO LIST

A CODE-ALONG

12. FETCHING DATA

EXTERNAL API

(20 MINUTES)

EXERCISE #12

API ADVENTURES IN A GALAXY FAR FAR AWAY

A CODE-ALONG

13. MULTITASKING

CONCURRENCY

GOROUTINES

- ▶ A **Goroutine** is a lightweight thread managed by the Go runtime
- ▶ Implemented by adding the **go** keyword before executing a function

```
4
5   func doSomethingSlow() {}
6
7   func main() {
8     go doSomethingSlow()
9   }
10
```

- ▶ Tells go to spin up a new thread to do that thing

(10 MINUTES)

EXERCISE #13

ADDING CONCURRENCY TO AN APP

A CODE-ALONG

ERROR WRAPPING

An error "e" can wrap another error "w" by providing an `Unwrap` method that returns `w`. Both `e` and `w` are available to programs, allowing `e` to provide additional context to `w` or to reinterpret it while still allowing programs to make decisions based on `w`.

**14. WE SHOULD DO
THIS AGAIN SOMETIME**

CONCLUSION

RESOURCES

- ▶ - [Offical Golang Docs]
 - ▶ (<https://golang.org/doc/>)
- ▶ - [How To Use Interfaces In Go]
 - ▶ (<https://jordanorelli.com/post/32665860244/how-to-use-interfaces-in-go>)
- ▶ - [Introducing Go]
 - ▶ (<http://shop.oreilly.com/product/0636920046516.do>), Caleb Doxsey, O'Reilly Publications
- ▶ - [Web Applications With Go]
 - ▶ (<https://blog.scottlogic.com/2017/02/28/building-a-web-app-with-go.html>)
- ▶ - [Go Language Programming Practical Basic Tutorial]
 - ▶ (<https://www.youtube.com/playlist?list=PLQVvva0QuDeF3hP0wQoSxpkqgRcgxMqX>)
- ▶ - [Star Wars API]
 - ▶ (<https://swapi.co/>)
- ▶ - My colleague Justin Holmes, and former colleagues Mike McCrary and Steven Bogacz for their patience with my endless questions.

CONCLUSION: CONTACT ME

CONTACT ME

- ▶ [Github](https://github.com/martensonbj): github.com/martensonbj
- ▶ [Twitter](https://twitter.com/martenson_bj): [@martenson_bj](https://twitter.com/martenson_bj)
- ▶ [Work](mailto:brenna.martenson@highwing.io): brenna.martenson@highwing.io
- ▶ [LinkedIn](https://www.linkedin.com/in/martensonbj): [linkedin/martensonbj](https://www.linkedin.com/in/martensonbj)

